CHAPTER

20

Configuring an NFS File Server

IN THIS CHAPTER

Getting NFS server software

Enabling and starting NFS

Exporting NFS directories

Setting security features for NFS

Mounting remote NFS shared directories

Instead of representing storage devices as drive letters (A, B, C, and so on), as they are in Microsoft operating systems, Linux systems invisibly connect filesystems from multiple hard disks, USB drives, CD-ROMs, and other local devices to form a single Linux filesystem. The Network File System (NFS) facility enables you to extend your Linux filesystem to connect filesystems on other computers to your local directory structure.

An NFS file server provides an easy way to share large amounts of data among the users and computers in an organization. An administrator of a Linux system that is configured to share its filesystems using NFS has to perform the following tasks to set up NFS:

1. **Set up the network**. NFS is typically used on private networks as opposed to public networks, such as the Internet.
2. **Start the NFS service.** Several service daemons need to start up and run to have a fully operational NFS service. In Fedora and Red Hat Enterprise Linux, you can start up the `nfs-server` service.
3. **Choose what to share from the server.** Decide which directories (folders) on your Linux NFS server to make available to other computers. You can choose any point in the filesystem and make all files and directories below that point accessible to other computers.
4. **Set up security on the server**. You can use several different security features to apply the level of security with which you are comfortable. *Mount-level security* enables you to restrict the computers that can mount a resource and, for those allowed to mount it, enables you to specify whether it can be mounted read/write or read-only. In NFS, user-level security is implemented by mapping users from the client systems to users on the NFS server (based on

UID and not username) so that they can rely on standard Linux read/write/execute permissions, file ownership, and group permissions to access and protect files.

5. **Mount the filesystem on the client.** Each client computer that is allowed access to the server's NFS shared filesystem can mount it anywhere the client chooses. For example, you may mount a filesystem from a computer called `oak` on the `/mnt/oak` directory in your local filesystem. After it is mounted, you can view the contents of that directory by typing `ls /mnt/oak`.

Although it is often used as a file server (or other type of server), Linux is a general-purpose operating system, so any Linux system can share, or export, filesystems as a server or use another computer's filesystems (mount) as a client. In fact, both Red Hat Enterprise Linux 8 and Fedora 30 Workstation include the `nfs-server` service in their default installations.

NOTE

A *filesystem* is usually a structure of files and directories that exists on a single device (such as a hard disk partition or CD-ROM). The term *Linux filesystem* refers to the entire directory structure (which may include filesystems from several disk partitions, NFS, or a variety of network resources), beginning from root (/) on a single computer. A shared directory in NFS may represent all or part of a computer's filesystem, which can be attached (from the shared directory down the directory tree) to another computer's filesystem.

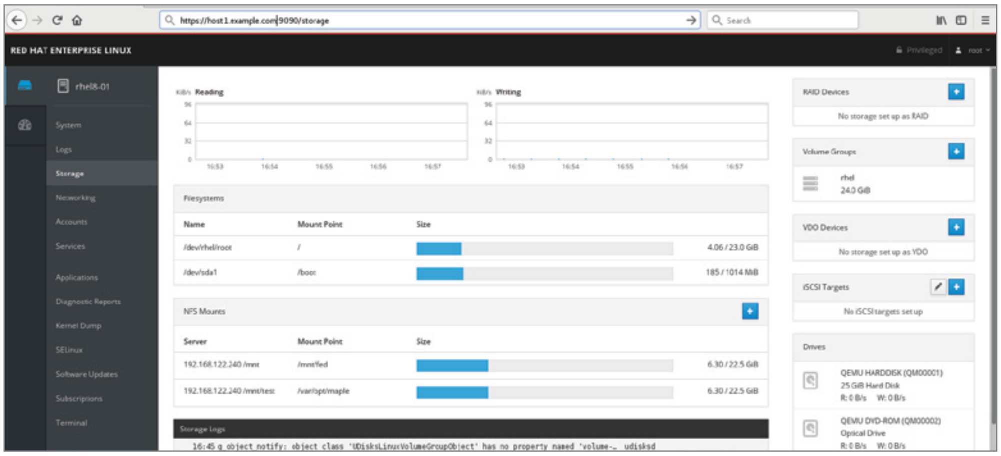
If you already have the NFS and Cockpit services running on your system, you can mount NFS shares and view mounted shares from the Cockpit Web UI. Here's how to do that:

1. Log in to your Cockpit interface (port 9090) through your web browser and select Storage. The URL to get to storage in the Cockpit service on your local system should be something like `https://host1.example.com:9090/storage`.
2. If there are mounted NFS shares on your system, they should appear under the NFS Mounts section. Figure 20.1 shows an example containing two mounted NFS shares.
3. To mount a remote NFS share, select the plus (+) sign on the NFS Mounts line. Fill in the address or hostname of the NFS server, the shared directory on the NFS share, and the point on the local file system where you will mount that share. Then select Add, as shown in Figure 20.2.

At this point, you should be able to access the content from the remote NFS share from the mount point on your local filesystem. By default, the NFS mount information is added to the `/etc/fstab` file, so the NFS share will be made available each time the system reboots. Now that you have seen the easy way to use NFS, the rest of the chapter describes how to use NFS from the ground up.

FIGURE 20.1

View NFS shares mounted locally using Cockpit Web UI



https://host1.example.com:9090/storage

RED HAT ENTERPRISE LINUX

rhel8-01

Filesystems

Name	Mount Point	Size
/dev/rhel/root	/	4.06 / 23.0 GiB
/dev/sda1	/boot	185 / 1014 MiB

NFS Mounts

Server	Mount Point	Size
192.168.122.240 /mnt	/mnt/fed	6.30 / 22.5 GiB
192.168.122.240 /mnt/test	/var/opt/maple	6.30 / 22.5 GiB

RAID Devices: No storage set up as RAID

Volume Groups: rhel 24.0 GiB

VDO Devices: No storage set up as VDO

iSCSI Targets: No iSCSI targets set up

Drives

QEMU HARDDISK (QM00001)
25 GiB Hard Disk
R: 0 B/s W: 0 B/s

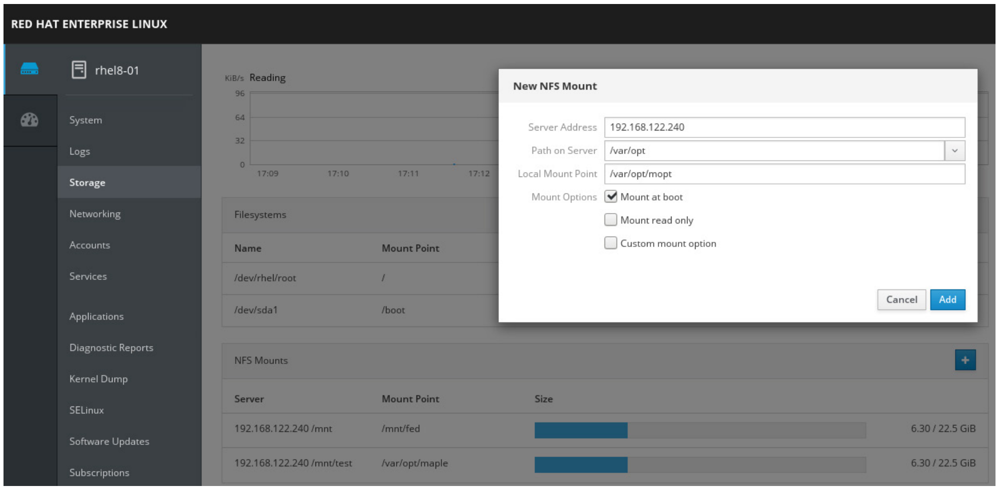
QEMU DVD-ROM (QM00002)
Optical Drive
R: 0 B/s W: 0 B/s

Storage Logs

```
16:45 g_object_notify: object class 'UDisksLinuxVolumeGroupObject' has no property named 'volume-… udisksd
```

FIGURE 20.2

Add a new NFS mount using Cockpit Web UI



RED HAT ENTERPRISE LINUX

rhel8-01

New NFS Mount

Server Address	192.168.122.240
Path on Server	/var/opt
Local Mount Point	/var/opt/mopt
Mount Options	✓ Mount at boot
	☐ Mount read only
	☐ Custom mount option

Filesystems

Name	Mount Point
/dev/rhel/root	/
/dev/sda1	/boot

NFS Mounts

Server	Mount Point	Size
192.168.122.240 /mnt	/mnt/fed	6.30 / 22.5 GiB
192.168.122.240 /mnt/test	/var/opt/maple	6.30 / 22.5 GiB

20

Installing an NFS Server

To run an NFS server, you need a set of kernel modules (which are delivered with the kernel itself) plus some user-level tools to configure the service, run daemon processes, and query the service in various ways.

For earlier releases of Fedora and RHEL, the components you need that are not already in the kernel can be added by installing the `nfs-utils` package. In RHEL 8 and Fedora 30, the required components are included in the following default installation:

```
# yum install nfs-utils
```

Besides a few documents in the `/usr/share/doc/nfs-utils` directory, most documentation in the `nfs-utils` package includes man pages for its various components. To see the list of documentation, type the following:

```
# rpm -qd nfs-utils | less
```

There are tools and man pages for both the NFS server side (for sharing a directory with others) and the client side (for mounting a remote NFS directory locally). To configure a server, you can refer to the exports man page (to set up the `/etc/exports` file to share your directories). The man page for the `exportfs` command describes how to share and view the list of directories that you share from the `/etc/exports` file. The `nfsd` man page describes the options that you can pass to the `rpc.nfsd` server daemon, which lets you do such things as run the server in debugging mode.

Man pages on the client side include the `mount.nfs` man page (to see what mount options you can use when mounting remote NFS directories on your local system). There is also an `nfsmount.conf` man page, which describes how to use the `/etc/nfsmount.conf` file to configure how your system behaves when you mount remote resources locally. The `showmount` man page describes how to use the `showmount` command to see what shared directories are available from NFS servers.

To find out more about the `nfs-utils` package, you can run the following commands to see information about the package, configuration files, and commands, respectively:

```
# rpm -qi nfs-utils
# rpm -qc nfs-utils
# rpm -ql nfs-utils | grep bin
```

Starting the NFS service

Starting the NFS server involves launching several service daemons. The basic NFS service in Fedora and RHEL 8 is called `nfs-server`. To start that service, enable it (so it starts each time your system boots) and check the status by running the following three commands:

```
# systemctl start nfs-server.service
# systemctl enable nfs-server.service
# systemctl status nfs-server.service
• nfs-server.service - NFS server and services
     Loaded: loaded (/lib/systemd/system/nfs-server.service; enabled
             vendor preset: disabled)
     Active: active (exited) since Mon 2019-9-02 15:15:11 EDT; 24s
ago
   Main PID: 7767 (code=exited, status=0/SUCCESS)
      Tasks: 0 (limit: 12244)
     Memory: 0B
     CGroup: /system.slice/nfs-server.service
```

You can see from the status that the `nfs-server` service is enabled and active. The NFS service also requires that the RPC service be running (`rpcbind`). The `nfs-server` service automatically starts the `rpcbind` service, if it is not already running.

In Red Hat Enterprise Linux 6, you need the `service` and `chkconfig` commands to check, start, and enable the NFS service (`nfs`). The following commands show the `nfs` service not running currently and disabled:

```
# service nfs status
rpc.svcgssd is stopped
rpc.mountd is stopped
nfsd is stopped
# chkconfig --list nfs
nfs  0:off  1:off  2:off  3:off  4:off  5:off  6:off
```

As mentioned earlier, the `rpcbind` service must be running for NFS to work. In RHEL 6, you could use the following commands to start and permanently enable both the `rpcbind` and `nfs` services:

```
# service rcpbind start
Starting rpcbind:                     [  OK  ]
# service nfs start
Starting NFS services:                [  OK  ]
Starting NFS quotas:                  [  OK  ]
Starting NFS daemon:                  [  OK  ]
Starting NFS mountd:                  [  OK  ]
# chkconfig rpcbind on
# chkconfig nfs on
```

After the service is running, the commands (`mount`, `exportfs`, and so on) and files (`/etc/exports`, `/etc/fstab`, and so on) for actually configuring NFS are basically the same on every Linux system. So, after you have NFS installed and running, just follow the instructions in this chapter to start using NFS.

Sharing NFS Filesystems

To share an NFS filesystem from your Linux system, you need to export it from the server system. Exporting is done in Linux by adding entries into the `/etc/exports` file. Each entry identifies a directory in your local filesystem that you want to share with other computers. The entry also identifies the other computers that can access the resource (or opens it to all computers) and includes other options that reflect permissions associated with the directory.

Remember that when you share a directory, you are sharing all files and subdirectories below that directory as well (by default). You need to be sure that you want to share everything in that directory structure. You can still restrict access within that directory structure in many ways; those are discussed later in this chapter.

Configuring the /etc/exports file

To make a directory from your Linux system available to other systems, you need to export that directory. Exporting is done on a permanent basis by adding information about an exported directory to the `/etc/exports` file.

Here's the format of the `/etc/exports` file:

```
Directory   Host(Options...)  Host(Options...)  # Comments
```

In this example, *`Directory`* is the name of the directory that you want to share, and Host indicates the client computer to which the sharing of this directory is restricted. *`Options`* can include a variety of options to define the security measures attached to the shared directory for the host. (You can repeat `Host` and `Option` pairs.) *`Comments`* are any optional comments that you want to add (following the # sign).

The `exports` man page (`man exports`) contains details about the syntax of the `/etc/exports` file. In particular, you can see the options that you can use to limit access and secure each shared directory.

As root user, you can use any text editor to configure `/etc/exports` to modify shared directory entries or add new ones. Here's an example of an `/etc/exports` file:

```
/cal    *.linuxtoys.net(rw)              # Company events
/pub    *(ro,insecure,all_squash)        # Public dir
/home   maple(rw,root_squash) spruce(rw,root_squash)
```

The `/cal` entry represents a directory that contains information about events related to the company. Any computer in the company's domain (`*.linuxtoys.net`) can mount that NFS share. Users can write files to the directory as well as read them (indicated by the `rw` option). The comment (`# Company events`) simply serves to remind you of what the directory contains.

The `/pub` entry represents a public directory. It allows any computer and user to read files from the directory (indicated by the `ro` option) but not to write files. The `insecure` option enables any computer, even one that doesn't use a secure NFS port, to access the directory. The `all _ squash` option causes all users (UIDs) and groups (GIDs) to be mapped to the `nobody` user (UID `65534`), giving them minimal permission to files and directories.

The /home entry enables a set of users to have the same /home directory on different computers. Suppose, for example, that you are sharing /home from a computer named oak. The computers named maple and spruce could each mount that directory on their own /home directories. If you gave all users the same username/UID on all machines, you could have the same /home/*user* directory available for each user, regardless of which computer they are logged into. The root _ squash is used to exclude the root user from another computer from having root privilege to the shared directory.

These are just examples; you can share any directories that you choose, including the entire filesystem (/). Of course, there are security implications of sharing the whole filesystem or sensitive parts of it (such as /etc). Security options that you can add to your /etc/exports file are described throughout the sections that follow.

Hostnames in /etc/exports

You can indicate in the /etc/exports file which host computers can have access to your shared directory. If you want to associate multiple hostnames or IP addresses with a particular shared directory, be sure to leave a space before each hostname. However, add no spaces between a hostname and its options. Here's an example:

```
/usr/local maple(rw) spruce(ro,root_squash)
```

Notice that there is a space after (rw) but none after maple. You can identify hosts in several ways:

Individual host Enter one or more TCP/IP hostnames or IP addresses. If the host is in your local domain, you can simply indicate the hostname. Otherwise, use the full host.domain format. These are valid ways to indicate individual host computers:

```
maple
maple.handsonhistory.com
10.0.0.11
```

IP network Allow access to all hosts from a particular network address by indicating a network number and its netmask, separated by a slash (/). Here are valid ways to designate network numbers:

```
10.0.0.0/255.0.0.0 172.16.0.0/255.255.0.0
192.168.18.0/255.255.255.0
192.168.18.0/24
```

TCP/IP domain Using wildcards, you can include all or some host computers from a particular domain level. Here are some valid uses of the asterisk and question mark wildcards:

```
*.handsonhistory.com
*craft.handsonhistory.com
???.handsonhistory.com
```

The first example matches all hosts in the `handsonhistory.com` domain. The second example matches `woodcraft`, `basketcraft`, or any other hostnames ending in `craft` in the `handsonhistory.com` domain. The final example matches any three-letter hostnames in the domain.

NIS groups You can allow access to hosts contained in an NIS group. To indicate an NIS group, precede the group name with an at (@) sign (for example, `@group`).

Access options in /etc/exports

You don't have to just give away your files and directories when you export a directory with NFS. In the options part of each entry in `/etc/exports`, you can add options that allow or limit access by setting read/write permission. These options, which are passed to NFS, are as follows:

`ro`: Client can mount this exported filesystem read-only. The default is to mount the filesystem read/write.

`rw`: Explicitly asks that a shared directory be shared with read/write permissions. (If the client chooses, it can still mount the directory as read-only.)

User mapping options in /etc/exports

In addition to options that define how permissions are handled generally, you can use options to set the permissions that specific users have to NFS shared filesystems.

One method that simplifies this process is to have each user with multiple user accounts have the same username and UID on each machine. This makes it easier to map users so they have the same permissions on a mounted filesystem as they do on files stored on their local hard disks. If that method is not convenient, user IDs can be mapped in many other ways. Here are some methods of setting user permissions and the `/etc/exports` option that you use for each method:

`root` user The client's root user is mapped by default into the `nobody` username (UID 65534). This prevents a client computer's root user from being able to change all files and directories in the shared filesystem. If you want the client's root user to have root permission on the server, use the `no_root_squash` option.

TIP

Keep in mind that even though root is squashed, the root user from the client can still become any other user account and access files for those user accounts on the server. So, be sure that you trust root with all of your user data before you share it read/write with a client.

`nfsnobody` or `nobody` user/group By using the `65534` user ID and group ID, you essentially create a user/group with permissions that do not allow access to files that belong to any real users on the server, unless those users open permission to everyone. However, files created by the `65534` user or group are available to anyone assigned as the 65534 user or group. To set all remote users to the 65534 user/group, use the all _ squash option.

The 65534 UIDs and GIDs are used to prevent the ID from running into a valid user or group ID. Using anonuid or anongid options, you can change the 65534 user or group, respectively. For example, anonuid=175 sets all anonymous users to UID 175, and anongid=300 sets the GID to 300. (Only the number is displayed when you list file permission unless you add entries with names to /etc/passwd and /etc/group for the new UIDs and GIDs.)

User mapping If a user has login accounts for a set of computers (and has the same ID), NFS, by default, maps that ID. This means that if the user named mike (UID 110) on maple has an account on pine (mike, UID 110), he can use his own remotely mounted files on either computer from either computer.

If a client user who is not set up on the server creates a file on the mounted NFS directory, the file is assigned to the remote client's UID and GID. (An ls -l on the server shows the UID of the owner.)

Exporting the shared filesystems

After you have added entries to your /etc/exports file, run the exportfs command to have those directories exported (made available to other computers on the network). Reboot your computer or restart the NFS service, and the exportfs command runs automatically to export your directories. If you want to export them immediately, run exportfs from the command line (as root).

TIP

Running the exportfs command after you change the exports file is a good idea. If any errors are in the file, exportfs identifies them for you.

Here's an example of the exportfs command:

```
# /usr/sbin/exportfs -a -r -v
exporting maple:/pub
exporting spruce:/pub
exporting maple:/home
exporting spruce:/home
exporting *:/mnt/win
```

The -a option indicates that all directories listed in /etc/exports should be exported. The -r resyncs all exports with the current /etc/exports file (disabling those exports no longer listed in the file). The -v option says to print verbose output. In this example, the /pub and /home directories from the local server are immediately available for mounting by those client computers that are named (maple and spruce). The /mnt/win directory is available to all client computers.

Securing Your NFS Server

The NFS facility was created at a time when encryption and other security measures were not routinely built into network services (such as remote login, file sharing, and remote execution). Therefore, NFS (even up through version 3) suffers from some rather glaring security issues.

NFS security issues made it an inappropriate facility to use over public networks and even made it difficult to use securely within an organization. These are some of the issues:

Remote `root` users Even with the default `root_squash` (which prevents root users from having root access to remote shares), the root user on any machine to which you share NFS directories can gain access to any other user account. Therefore, if you are doing something like sharing home directories with read/write permission, the root user on any box to which you are sharing has complete access to the contents of those home directories.

Unencrypted communications Because NFS traffic is unencrypted, anyone sniffing your network can see the data that is being transferred.

User mapping Default permissions to NFS shares are mapped by user ID. So, for example, a user with UID 1000 on an NFS client has access to files owned by UID 1000 on the NFS server. This is regardless of the usernames used.

Filesystem structure exposed Up to NFSv3, if you shared a directory over NFS, you exposed the location of that directory on the server's filesystem. (In other words, if you shared the `/var/stuff` directory, clients would know that `/var/stuff` was its exact location on your server).

That's the bad news. The good news is that most of these issues are addressed in NFSv4 but require some extra configuration. By integrating Kerberos support, NFSv4 lets you configure user access based on each user obtaining a Kerberos ticket. For you, the extra work is configuring a Kerberos server. As for exposing NFS share locations, with NFSv4 you can bind shared directories to an `/exports` directory, so when they are shared, the exact location of those directories is not exposed.

Visit `https://help.ubuntu.com/community/NFSv4Howto` for details on NFSv4 features in Ubuntu.

As for standard Linux security features associated with NFS, `iptables` firewalls, TCP wrappers, and SELinux can all play a role in securing and providing access to your NFS server from remote clients. In particular, getting firewall features working with NFS can be particularly challenging. These security features are described in the sections that follow.

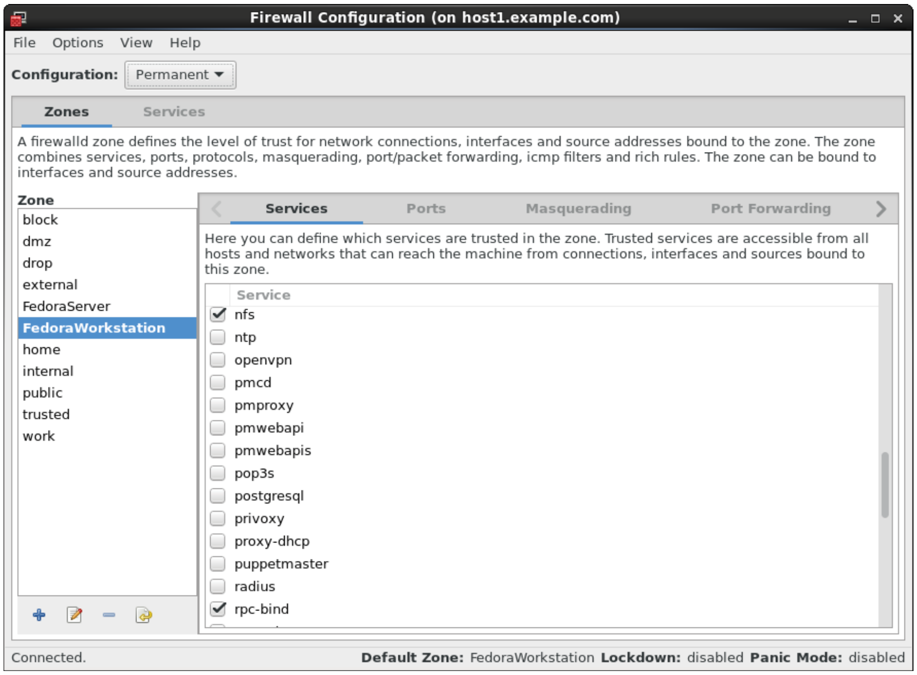
Opening up your firewall for NFS

The NFS service relies on several different service daemons for normal operation, with most of these daemons listening on different ports for access. For the default NFSv4 used in Fedora, TCP and UDP ports 2049 (`nfs`) and 111 (`rpcbind`) must be open for an NFS server to perform properly. The server must also open TCP and UDP ports 20048 for the `showmount` command to be able to query available NFS shared directories from `rpc.mountd` on the server.

For RHEL 8, Fedora 30, and other systems that use the `firewalld` service, you can use the Firewall Configuration window (`yum install firewall-config`) to open the firewall for your NFS service. Type `firewall-config`, then make sure that mountd, nfs, and rpc-bind are checked in the window to open the appropriate ports to allow access to your NFS service. Figure 20.3 shows an example of this window:

FIGURE 20.3

Use the Firewall Configuration window to open your firewall to allow access to the NFS service.



Configuration: Permanent

A firewalld zone defines the level of trust for network connections, interfaces and source addresses bound to the zone. The zone combines services, ports, protocols, masquerading, port/packet forwarding, icmp filters and rich rules. The zone can be bound to interfaces and source addresses.

Zone
block
dmz
drop
external
FedoraServer
FedoraWorkstation
home
internal
public
trusted
work

Here you can define which services are trusted in the zone. Trusted services are accessible from all hosts and networks that can reach the machine from connections, interfaces and sources bound to this zone.

	Service
✓	nfs
	ntp
	openvpn
	pmcd
	pmproxy
	pmwebapi
	pmwebapis
	pop3s
	postgresql
	privoxy
	proxy-dhcp
	puppetmaster
	radius
✓	rpc-bind

Connected. **Default Zone:** FedoraWorkstation **Lockdown:** disabled **Panic Mode:** disabled

For RHEL 6 and other systems that use `iptables` service directly (prior to `firewalld` being added), to open ports on the NFS server's firewall, make sure `iptables` is enabled and started with firewall rules similar to the following added to the `/etc/sysconfig/iptables` file:

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 111 -j ACCEPT
-A INPUT -m state --state NEW -m udp -p udp --dport 111 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 2049 -j ACCEPT
```

```
-A INPUT -m state --state NEW -m udp -p udp --dport 2049 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 20048 -j ACCEPT
-A INPUT -m state --state NEW -m udp -p udp --dport 20048 -j ACCEPT
```

In Red Hat Enterprise Linux 6.*x* and earlier, the firewall issue is a bit more complex. The problem, as it relates to firewalls, is that several different services are associated with NFS that listen on different ports, and those ports are assigned randomly. To get around that problem, you need to lock down the port numbers those services use and open the firewall so that those ports are accessible.

To make the process of locking down NFS server ports easier, entries in the `/etc/sysconfig/nfs` file can be added to assign specific port numbers to services. The following are examples of options in the `/etc/sysconfig/nfs` file with static port numbers set:

```
RQUOTAD_PORT=49001
LOCKD_TCPPORT=49002
LOCKD_UDPPORT=49003
MOUNTD_PORT=49004
STATD_PORT=49005
STATD_OUTGOING_PORT=49006
RDMA_PORT=49007
```

With those ports set, I restarted the `nfs` service (`service nfs restart`). Using the `netstat` command, you can see the resulting processes that are listening on those assigned ports:

```
tcp  0  0 0.0.0.0:49001   0.0.0.0:*    LISTEN    4682/rpc.rquotad
tcp  0  0 0.0.0.0:49002   0.0.0.0:*    LISTEN    -
tcp  0  0 0.0.0.0:49004   0.0.0.0:*    LISTEN    4698/rpc.mountd
tcp  0  0 :::49002        :::*         LISTEN    -
tcp  0  0 :::49004        :::*         LISTEN    4698/rpc.mountd
udp  0  0 0.0.0.0:49001   0.0.0.0:*              4682/rpc.rquotad
udp  0  0 0.0.0.0:49003   0.0.0.0:*              -
udp  0  0 0.0.0.0:49004   0.0.0.0:*              4698/rpc.mountd
udp  0  0 :::49003        :::*                   -
udp  0  0 :::49004        :::*                   4698/rpc.mountd
```

With those port numbers set and being used by the various services, you can now add `iptables` rules, as you did with ports 2049 and 111 for the basic NFS service.

Allowing NFS access in TCP wrappers

For services such as `vsftpd` and `sshd`, TCP wrappers in Linux enable you to add information to `/etc/hosts.allow` and `/etc/hosts.deny` files to indicate which hosts can or cannot access the service. Although the `nfsd` server daemon itself is not enabled for TCP wrappers, the `rpcbind` service is.

For NFSv3 and earlier versions, simply adding a line such as the following to the `/etc/hosts.deny` file would deny access to the `rpcbind` service, but it would also deny access to your NFS service:

```
rpcbind: ALL
```

For servers running NFSv4 by default, however, the `rpcbind: ALL` line just shown prevents outside hosts from getting information about RPC services (such as NFS) using commands like `showmount`. However, it does not prevent you from mounting an NFS shared directory.

Configuring SELinux for your NFS server

With SELinux set to permissive or disabled, it does not block access to the NFS service. In enforcing mode, however, you should understand a few SELinux Booleans. To check the state of SELinux on your system, enter the following:

```
# getenforce
Enforcing
# grep ^SELINUX= /etc/sysconfig/selinux
SELINUX=enforcing
```

If your system is in enforcing mode, as it is here, check the `nfs _ selinux` man page for information about SELinux settings that can impact the operation of your `vsftpd` service. Here are a few SELinux file contexts associated with NFS that you might need to know about:

`nfs_export_all_ro`: With this Boolean set to on, SELinux allows you to share files with read-only permission using NFS. NFS read-only file sharing is allowed with this on regardless of the SELinux file context set on the shared files and directories.

`nfs_export_all_rw`: With this Boolean set to on, SELinux allows you to share files with read/write permission using NFS. As with the previous Boolean, this works regardless of the file context set on the shared files and directories.

`use_nfs_home_dirs`: To allow the NFS server to share your home directories via NFS, set this Boolean to on.

Of the Booleans just described, the first two are on by default. The `use_nfs_home_dirs` Boolean is off. To turn on the `use_nfs_home_dirs` directory, you could type the following:

```
# setsebool -P use_nfs_home_dirs on
```

You can ignore all of the Booleans related to NFS file sharing, however, by changing the file contexts on the files and directories you want to share via NFS. The `public_content_t` and `public_content_rw_t` file contexts can be set on any directory that you want to share via NFS (or other file share protocols, such as HTTP, FTP, and others, for that matter). For example, to set the rule to allow the `/whatever` directory and its subdirectories to be shared read/write via NFS, and then to apply that rule, enter the following:

```
# semanage fcontext -a -t public_content_rw_t "/whatever(/.*)?"
# restorecon -F -R -v /whatever
```

If you wanted to allow users just to be able to read files from a directory, but not write to it, you could assign the `public_content_t` file context to the directory instead.

Using NFS Filesystems

After a server exports a directory over the network using NFS, a client computer connects that directory to its own filesystem using the `mount` command. That's the same command used to mount filesystems from local hard disks, DVDs, and USB drives, but with slightly different options.

The `mount` command enables a client to mount NFS directories added to the `/etc/fstab` file automatically, just as it does with local disks. NFS directories can also be added to the `/etc/fstab` file in such a way that they are not automatically mounted (so you can mount them manually when you choose). With a `noauto` option, an NFS directory listed in `/etc/fstab` is inactive until the `mount` command is used, after the system is up and running, to mount the filesystem.

In addition to the `/etc/fstab` file, you can set mount options using the `/etc/nfsmount.conf` file. Within that file, you can set mount options that apply to any NFS directory you mount or only those associated with specific mount points or NFS servers.

Before you set about mounting NFS shared directories, however, you probably want to check out what shared directories are available via NFS using the `showmount` command.

Viewing NFS shares

From a client Linux system, you can use the `showmount` command to see what shared directories are available from a selected computer, such as in this example:

```
$ showmount -e server.example.com
/export/myshare client.example.com
/mnt/public     *
```

The `showmount` output shows that the shared directory named `/export/myshare` is available only to the host `client.example.com`. The `/mnt/public` shared directory, however, is available to anyone.

Manually mounting an NFS filesystem

After you know that the directory from a computer on your network has been exported (that is, made available for mounting), you can mount that directory manually using the `mount` command. This is a good way to make sure that it is available and working before you set it up to mount permanently. The following is an example of mounting the `/stuff` directory from a computer named `maple` on your local computer:

```
# mkdir /mnt/maple
# mount maple:/stuff /mnt/maple
```

The first command (mkdir) creates the mount point directory. (/mnt is a common place to put temporarily mounted disks and NFS filesystems.) The mount command identifies the remote computer and shared filesystem, separated by a colon (maple:/stuff), and the local mount point directory (/mnt/maple) follows.

NOTE

If the mount fails, make sure that the NFS service is running on the server and that the server's firewall rules don't deny access to the service. From the server, type **ps ax | grep nfsd** to see a list of nfsd server processes. If you don't see the list, try to start your NFS daemons as described earlier in this chapter. To view your firewall rules, type **iptables -vnL**. By default, the nfsd daemon listens for NFS requests on port number 2049. Your firewall must accept udp requests on ports 2049 (nfs) and 111 (rpc). In Red Hat Enterprise Linux 6 and earlier versions of Fedora, you may need to set static ports for related services and then open ports for those services in the firewall. Refer to the section "Securing Your NFS Server" earlier in this chapter to review how to overcome these security issues.

To ensure that the NFS mount occurred, type **mount -t nfs4**. This command lists all mounted NFS filesystems. Here is an example of the mount command and its output (with filesystems not pertinent to this discussion edited out):

```
# mount -t nfs4
192.168.122.240:/mnt on /mnt/fed type nfs4
(rw,relatime,vers=4.2,rsize=262144,wsize=262144,namlen=255,hard,
proto=tcp,timeo=600,retrans=2,sec=sys,clientaddr=192.168.122.63,
local_lock=none,addr=192.168.122.240)
```

The output from the mount -t nfs4 command shows only those filesystems mounted from NFS file servers. The NFS filesystem is the /mnt directory from 192.168.122.240 (192.168.122.240:/mnt). It is mounted on /mnt/fed, and its mount type is nfs4. The filesystem was mounted read/write (rw), and the IP address of maple is 192.168.122.240 (addr=192.168.122.240). Many other settings related to the mount are shown as well, such as the read and write sizes of packets and the NFS version number.

The mount operation just shown temporarily mounts an NFS filesystem on the local system. The next section describes how to make the mount more permanent (using the /etc/fstab file) and how to select various options for NFS mounts.

Mounting an NFS filesystem at boot time

To set up an NFS filesystem to mount automatically on a specified mount point each time you start your Linux system, you need to add an entry for that NFS filesystem to the /etc/fstab file. That file contains information about all different kinds of mounted (and available to be mounted) filesystems for your system.

Here's the format for adding an NFS filesystem to your local system:

```
host:directory    mountpoint    nfs    options    0    0
```

The first item (*host:directory*) identifies the NFS server computer and shared directory. *mountpoint* is the local mount point on which the NFS directory is mounted. It is followed by the filesystem type (nfs). Any options related to the mount appear next in a comma-separated list. (The last two zeros configure the system not to dump the contents of the filesystem and not to run fsck on the filesystem.)

The following are examples of NFS entries in /etc/fstab:

```
maple:/stuff   /mnt/maple nfs   bg,rsize=8192,wsize=8192  0 0
oak:/apps      /oak/apps  nfs   noauto,ro                 0 0
```

In the first example, the remote directory /stuff from the computer named maple (maple:/stuff) is mounted on the local directory /mnt/maple (the local directory must already exist). If the mount fails because the share is unavailable, the bg causes the mount attempt to go into the background and retry again later.

The filesystem type is nfs, and read (rsize) and write (wsize) buffer sizes (discussed in the section "Using mount options," later in this chapter) are set at 8192 to speed data transfer associated with this connection. In the second example, the remote directory is /apps on the computer named oak. It is set up as an NFS filesystem (nfs) that can be mounted on the /oak/apps directory locally. This filesystem is not mounted automatically (noauto), however, and it can be mounted only as read-only (ro) using the mount command after the system is already running.

TIP

The default is to mount an NFS filesystem as read/write. However, the default for exporting a filesystem is read-only. If you are unable to write to an NFS filesystem, check that it was exported as read/write from the server.

Mounting noauto filesystems

Your /etc/fstab file may also contain devices for other filesystems that are not mounted automatically. For example, you might have multiple disk partitions on your hard disk or an NFS shared filesystem that you want to mount only occasionally. A noauto filesystem can be mounted manually. The advantage is that when you type the mount command, you can type less information and have the rest filled in by the contents of the /etc/fstab file. So, for example, you could type

```
# mount /oak/apps
```

With this command, mount knows to check the /etc/fstab file to get the filesystem to mount (oak:/apps), the filesystem type (nfs), and the options to use with the mount (in this case ro, for read-only). Instead of typing the local mount point (/oak/apps), you could have typed the remote filesystem name (oak:/apps) and had other information filled in.

TIP
When naming mount points, including the name of the remote NFS server in that name can help you remember where the files are actually being stored. This may not be possible if you are sharing home directories (`/home`) or mail directories (`/var/spool/mail`). For example, you might mount a filesystem from a machine called `duck` on the directory `/mnt/duck`.

Using mount options

You can add several `mount` options to the `/etc/fstab` file (or to a `mount` command line itself) to influence how the filesystem is mounted. When you add options to `/etc/fstab`, they must be separated by commas. For example, here the `noauto`, `ro`, and `hard` options are used when `oak:/apps` is mounted:

```
oak:/apps    /oak/apps  nfs   noauto,ro,hard    0 0
```

The following are some options that are valuable for mounting NFS filesystems. You can read about these and other NFS mount options you can put in the `/etc/fstab` file from the `nfs` man page (`man 5 nfs`):

`hard` If this option is used and the NFS server disconnects or goes down while a process is waiting to access it, the process hangs until the server comes back up. This is helpful if it is critical that the data with which you are working stay in sync with the programs that are accessing it. (This is the default behavior.)

`soft` If the NFS server disconnects or goes down, a process trying to access data from the server times out after a set period when this option is on. An input/output error is delivered to the process trying to access the NFS server.

`rsize` This is the size of the blocks of data (in bytes) that the NFS client will request be used when it is reading data from an NFS server. The default is 1024. Using a larger number (such as 8192) gets you better performance on a network that is fast (such as a LAN) and is relatively error-free (that is, one that doesn't have lots of noise or collisions).

`wsize` This is the size of the blocks of data (in bytes) that the NFS client will request to be used when it is writing data to an NFS server. The default is 1024. Performance issues are the same as with the `rsize` option.

`timeo=#` This sets the time after an RPC time-out occurs that a second transmission is made, where # represents a number in tenths of a second. The default value is seven-tenths of a second. Each successive time-out causes the time-out value to be doubled (up to 60 seconds maximum). Increase this value if you believe that time-outs are occurring because of slow response from the server or a slow network.

`retrans=#` This sets the number of minor time-outs and retransmissions that need to happen before a major time-out occurs.

retry=# This sets how many minutes to continue to retry failed mount requests, where # is replaced by the number of minutes to retry. The default is 10,000 minutes (which is about one week).

bg If the first mount attempt times out, try all subsequent mounts in the background. This option is very valuable if you are mounting a slow or sporadically available NFS filesystem. When you place mount requests in the background, your system can continue to mount other filesystems instead of waiting for the current one to complete.

NOTE

If a nested mount point is missing, a time-out to allow for the needed mount point to be added occurs. For example, if you mount /usr/trip and /usr/trip/extra as NFS filesystems and /usr/trip is not yet mounted when /usr/trip/extra tries to mount, /usr/trip/extra times out. If you're lucky, /usr/trip comes up and /usr/trip/extra mounts on the next retry.

fg If the first mount attempt times out, try subsequent mounts in the foreground. This is the default behavior. Use this option if it is imperative that the mount be successful before continuing (for example, if you were mounting /usr).

Not all NFS mount options need to go into the /etc/fstab file. On the client side, the /etc/nfsmount.conf file can be configured for Mount, Server, and Global sections. In the Mount section, you can indicate which mount options are used when an NFS filesystem is mounted to a particular mount point. The Server section lets you add options to any NFS filesystem mounted from a particular NFS server. Global options apply to all NFS mounts from this client.

The following entry in the /etc/nfsmount.conf file sets a 32KB read and write block size for any NFS directories mounted from the system named thunder.example.com:

```
[ Server "thunder.example.com" ]
  rsize=32k
  wsize=32k
```

To set default options for all NFS mounts for your systems, you can uncomment the NFSMount _ Global _ Options block. In that block, you can set such things as protocols and NFS versions as well as transmission rates and retry settings. Here is an example of an NFSMount _ Global _ Options block:

```
[ NFSMount_Global_Options ]
# This sets the default version to NFS 4
Defaultvers=4
# Sets the number of times a request will be retried before
# generating a timeout
Retrans=2
# Sets the number of minutes before retrying a failed
# mount to 2 minutes
Retry=2
```

In the example just shown, the default NFS version is 4. Data is retransmitted twice (2) before generating a time-out. The wait time is 2 minutes before retrying a failed transmission. You can override any of these default values by adding mount options to the `/etc/fstab` or to the mount command line when the NFS directory is mounted.

Using autofs to mount NFS filesystems on demand

Improvements to autodetecting and mounting removable devices have meant that you can simply insert or plug in those devices to have them detected, mounted, and displayed. However, to make the process of detecting and mounting remote NFS filesystems more automatic, you still need to use a facility such as `autofs` (short for *automatically mounted filesystems*).

The `autofs` facility mounts network filesystems on demand when someone tries to use the filesystems. With the `autofs` facility configured and turned on, you can cause any available NFS shared directories to mount on demand. To use the `autofs` facility, you need to have the `autofs` package installed. (For Fedora and RHEL, you can type **`yum install autofs`** or for Ubuntu or Debian **`apt-get install autofs`** to install the package from the network.)

Automounting to the /net directory

With `autofs` enabled, if you know the hostname and directory being shared by another host computer, simply change (`cd`) to the `autofs` mount directory (`/net` or `/var/autofs` by default). This causes the shared resource to be automatically mounted and made accessible to you.

The following steps explain how to turn on the `autofs` facility in Fedora or RHEL:

1. **In Fedora or RHEL, as root user from a Terminal window, open the `/etc/auto.master` file and look for the following line:**

   ```
   /net    -hosts
   ```

   This causes the `/net` directory to act as the mount point for the NFS shared directories that you want to access on the network. (If there is a comment character at the beginning of that line, remove it.)
2. **To start the `autofs` service in a Fedora 30, RHEL 7, or later system, type the following as root user:**

   ```
   # systemctl start autofs.service
   ```
3. **On a Fedora 30, RHEL 7, or later system, set up the `autofs` service to restart every time you boot your system:**

   ```
   # systemctl enable autofs
   ```

Believe it or not, that's all you have to do. If you have a network connection to the NFS servers from which you want to share directories, try to access a shared NFS directory. For example, if you know that the /usr/local/share directory is being shared from the computer on your network named shuttle, you can do the following:

```
$ cd /net/shuttle/
```

If that computer has any shared directories that are available to you, you can successfully change to that directory.

You also can type the following:

```
$ ls
usr
```

You should be able to see that the usr directory is part of the path to a shared directory. If there were shared directories from other top-level directories (such as /var or /tmp), you would see those. Of course, seeing any of those directories depends on how security is set up on the server.

Try going straight to the shared directory, as shown in this example:

```
$ cd /net/shuttle/usr/local/share
$ ls
info man music television
```

At this point, the ls should reveal the contents of the /usr/local/share directory on the computer named shuttle. What you can do with that content depends on how it was configured for sharing by the server.

This can be a bit disconcerting because you don't see any files or directories until you actually try to use them, such as changing to a network-mounted directory. The ls command, for example, doesn't show anything under a network-mounted directory until the directory is mounted, which may lead to a sometimes-it's-there-and-sometimes-it's-not impression. Just change to a network-mounted directory, or access a file on such a directory, and autofs takes care of the rest.

In the example shown, the hostname shuttle is used. However, you can use any name or IP address that identifies the location of the NFS server computer. For example, instead of shuttle, you might have used shuttle.example.com or an IP address such as 192.168.0.122.

Automounting home directories

Instead of just mounting an NFS filesystem under the /net directory, you might want to configure autofs to mount a specific NFS directory in a specific location. For example, you could configure a user's home directory from a centralized server that could be automounted from a different machine when a user logs in. Likewise, you could use a central authentication mechanism, such as LDAP (as described in Chapter 11, "Managing User Accounts"), to offer centralized user accounts.

The following procedure illustrates how to set up a user account on an NFS server and share the home directory of a user named `joe` from that server so that it can be automounted when `joe` logs into a different computer. In this example, instead of using a central authentication server, matching accounts are created on each system.

1. On the NFS server (`mynfs.example.com`) that provides a centralized user home directory for the user named `joe`, create a user account for `joe` with a home directory of `/home/shared/joe` as its name. Also find `joe`'s user ID number from the `/etc/passwd` file (third field) so that you can match it when you set up a user account for `joe` on another system.

   ```
   # mkdir /home/shared
   # useradd -c "Joe Smith" -d /home/shared/joe joe
   # grep joe /etc/passwd
   joe:x:1000:1000:Joe Smith:/home/shared/joe:/bin/bash
   ```

2. On the NFS server, export the `/home/shared/` directory to any system on your local network (I use 192.168.0.* here), so that you can share the home directory for `joe` and any other users you create by adding this line to the `/etc/exports` file:

   ```
   # /etc/exports file to share directories under /home/shared
   # only to other systems on the 192.168.0.0/24 network:
   /home/shared 192.168.0.*(rw,insecure)
   ```

NOTE

In the exports file example above, the `insecure` option allows clients to use ports above port 1024 to make mount requests. Some NFS clients require this because they do not have access to NFS-reserved ports.

3. On the NFS server, restart the `nfs-server` service, or if it is already running, you can simply export the shared directory as follows:

   ```
   # exportfs -a -r -v
   ```

4. On the NFS server, make sure that the appropriate ports are open on the firewall. See the section "Securing Your NFS Server" earlier in this chapter for details.
5. On the NFS client system, add an entry to the `/etc/auto.master` file that identifies the mount point where you want the remote NFS directory to be mounted and a file (of your choosing) where you will identify the location of the remote NFS directory. I added this entry to the `auto.master` file:

   ```
   /home/remote /etc/auto.joe
   ```

6. On the NFS client system, add an entry to the file you just noted (`/etc/auto.joe` is what I used) that contains an entry like the following:

   ```
   joe       -rw     mynfs.example.com:/home/shared/joe
   ```

7. On the NFS client system, restart the `autofs` service:

   ```
   # systemctl restart autofs.service
   ```

8. On the NFS client system, create a user named joe using the useradd command. For that command line, you need to get the UID for joe on the server (507 in this example) so that joe on the client system owns the files from joe's NFS home directory. When you run the following command, the joe user account is created, but you will see an error message stating that the home directory already exists (which is correct):

```
# useradd -u 507 -c "Joe Smith" -d /home/remote/joe joe
# passwd joe
Changing password for user joe.
New password: ********
Retype new password: ********
```

9. On the NFS client system, log in as joe. If everything is working properly, when joe logs in and tries to access his home directory (/home/remote/joe), the directory /home/share/joe should be mounted from the mynfs.example.com server. The NFS directory was both shared and mounted as read/write with ownership to UID 507 (joe on both systems), so the user joe on the local system should be able to add, delete, change, and view files in that directory.

After joe logs off (actually, when he stops accessing the directory) for a time-out period (10 minutes, by default), the directory is unmounted.

Unmounting NFS filesystems

After an NFS filesystem is mounted, unmounting it is simple. You use the umount command with either the local mount point or the remote filesystem name. For example, here are two ways that you could unmount maple:/stuff from the local directory /mnt/maple:

```
# umount maple:/stuff
# umount /mnt/maple
```

Either form works. If maple:/stuff is mounted automatically (from a listing in /etc/fstab), the directory is remounted the next time you boot Linux. If it was a temporary mount (or listed as noauto in /etc/fstab), it isn't remounted at boot time.

TIP
The command is `umount`, not unmount. This is easy to get wrong.

If you get the message device is busy when you try to unmount a filesystem, it means that the unmount failed because the filesystem is being accessed. Most likely, one of the directories in the NFS filesystem is the current directory for your shell (or the shell of someone else on your system). The other possibility is that a command is holding a file open in the NFS filesystem (such as a text editor). Check your Terminal windows and other shells, and then `cd` out of the directory if you are in it, or just close the Terminal windows.

If an NFS filesystem doesn't unmount, you can force it (`umount -f /mnt/maple`) or unmount and clean up later (`umount -l /mnt/maple`). The `-l` option is usually the better choice because a forced unmount can disrupt a file modification that is in progress. Another alternative is to run `fuser -v` *`mountpoint`* to see what users are holding your mounted NFS share open and then `fuser -k` *`mountpoint`* to kill all of those processes.

Summary

Network File System (NFS) is one of the oldest computer file sharing products in existence today. It is still the most popular for sharing directories of files between UNIX and Linux systems. NFS allows servers to designate specific directories to make available to designated hosts and then allows client systems to connect to those directories by mounting them locally.

NFS can be secured using firewall (`iptables`) rules, TCP wrappers (to allow and deny host access), and SELinux (to confine how file sharing protocols can share NFS resources). Although NFS was inherently insecure when it was created (data is shared unencrypted and user access is fairly open), features in NFS version 4 have helped improve the overall security of NFS.

This NFS chapter is the last of the book's server chapters. Chapter 21, "Troubleshooting Linux," covers a wide range of desktop and server topics as it helps you understand techniques for troubleshooting your Linux system.

Exercises

Exercises in this section take you through tasks related to configuring and using an NFS server in Linux. If possible, have two Linux systems available that are connected on a local network. One of those Linux systems will act as an NFS server while the other will be an NFS client.

To get the most from these exercises, I recommend that you don't use a Linux server that has NFS already up and running. You can't do all of the exercises here without disrupting an NFS service that is already running and sharing resources.

See Appendix B for suggested solutions.

1. On the Linux system you want to use as an NFS server, install the packages needed to configure an NFS service.
2. On the NFS server, list the documentation files that come in the package that provides the NFS server software.

3. On the NFS server, determine the name of the NFS service and start it.
4. On the NFS server, check the status of the NFS service you just started.
5. On the NFS server, create the `/var/mystuff` directory and share it from your NFS server with the following attributes: available to everyone, read-only, and the root user on the client has root access to the share.
6. On the NFS server, make sure that the share you created is accessible to all hosts by opening TCP wrappers, `iptables`, and SELinux.
7. On a second Linux system (NFS client), view the shares available from the NFS server. (If you don't have a second system, you can do this from the same system.) If you do not see the shared NFS directory, go back to the previous question and try again.
8. On the NFS client, create a directory called `/var/remote` and temporarily mount the `/var/mystuff` directory from the NFS server on that mount point.
9. On the NFS client, unmount `/var/remote`, add an entry so that the same mount is done automatically when you reboot (with a `bg` mount option), and test that the entry you created is working properly.
10. From the NFS server, copy some files to the `/var/mystuff` directory. From the NFS client, make sure that you can see the files just added to that directory and make sure that you can't write files to that directory from the client.